DPGS Graph Summarization Preserves Community Structure

L Durbeck
Virginia Tech
Dept. of ECE
ldurbeck @ vt.edu

Peter Athanas
Virginia Tech
Dept. of ECE
athanas @ vt.edu

***Abstract*—Community detection, or clustering, is an NP-hard problem for which computationally efficient approximations have long been sought. Methods have sought a balance between accuracy, simplicity, latency, and memory usage. In this paper, we assess whether a recently developed graph summarization technique preserves the underlying community structure of the graph. The technique is Degree-Preserving Graph Summarization (DPGS). It lossily encodes the graph using fewer nodes and edges but does not assume a uniform distribution of node degrees, as is common in most Minimum Description Length-based GS methods. Instead, it strives to improve the reconstruction model for graphs with realistic skewed degrees using the configuration model. We show that the supernodes produced by DPGS capture the latent communities within the graph for a series of graphs with known community structure, while reducing the graph description length by 25%.**

***Index Terms*—Computation (stat.CO); Graph partition; graph sparsification; graph summarization; MDL; spectral graph theory; LOBPCG; spectral graph partitioning; stochastic block models; community detection; DARPA/MIT Graph Challenge**

I. Introduction

Graphs can be used to represent the relationships between entities. For example, a social networking site such as Facebook uses a large graph to represent the social networks of its users, and then uses this information to alert only certain people when a certain user posts new content. A graph partition is a more compact encoding of the data, in which the community-level structure of the graph is preserved, with a several-fold reduction in the number of edges used to describe the graph. A closely related problem is graph summarization, for which a more compact encoding of the graph is sought—generally without the requirement that the encoding preserves the community structure of the original graph. In this paper we examine a lossy graph summarization technique from the general class of Minimum Description Length (MDL) approaches that appears to apply naturally to the separate problem of community detection. If this were so, this could be a technique for providing a compressed but high-fidelity encoding of the community structure, along with information used to reconstruct the original graph.

Graph partitioning is among NP-hard problems for which exact solutions are too expensive to obtain because no polynomial-time solution is known; instead, heuristics are used to find clusters that approximate the optimal solution sufficiently well. Modern phenomenal data production rates from the digitization of most aspects of human life make it desirable to have a simple, fast and accurate graph partitioner along with convenient tools for graph analysis.

The DARPA/MIT Graph Challenge (DGC) was envisioned as a way to accelerate progress in graph structure assessments such as graph isomorphism and graph partition [8]. One of its contributions is a set of stochastic block model-based (SBM) graphs, some with real-world graph embeddings, along with vertex group labels, with which researchers can develop new algorithms and validate results. This also facilitates comparisons among competing approaches in terms of their computational efficiency versus the accuracy of their results.

The Streaming Stochastic Block Graph Challenge (SGC) provides the above graph dataset, which contains forty synthetic-real hybrid SBM graphs with power-law degree distributions that reflect real-world graphs and known community labels. The dataset includes both static and streaming datasets for the graphs. It also includes a graph generator; a baseline algorithm; scoring functions against which researchers can benchmark their progress; and a reference implementation of the baseline in Python with which to begin.

The graph generator implements Karrer and New-

in *IEEE High Performance Extreme Computing Conference HPEC 2021*,

man's classic stochastic blockmodel [10] broadened to represent a greater range of graphs using a hybrid mixed-membership model (HMMB) of Kao [9]. The graphs in the datasets provided for the SGC also include embeddings of actual graphs that further increase their realism.

This paper reports on efforts to apply the rigors of comparison promoted by the SGC another step further, into the closely associated research topic of graph summarization—an area that is advancing rapidly because of its direct applicability to a hot topic, graph neural network (GNN) sparsification to accelerate graph mining.

This paper reports on a rigorous side-by-side comparison of three candidate approaches and implementations:

- DPGS, a recent advancement in graph summarization that appears suitable [23];
- Peixoto, the SGC baseline based on Peixoto [8], [18]; and
- LOBPCG, the fastest and most accurate sequential approach demonstrated to date in SGC, based on the LOBPCG spectral method [5], [24].

The paper is organized as follows. The Background section briefly describes DPGS. The Related Work sections outlines the relevant research areas and provides some motivation for the research; it also discusses other approaches. The Approach section outlines the research question and our approach to answering it. The Methods section describes the experimental setup. The Experiments and Results section presents specific evaluations and characterizes the three algorithms relative to one another. Discussion and Future Work summarizes the work and suggests some specific ways in which it could be extended.

II. Background

Degree-Preserving Graph Summarization algorithm (DPGS) by Zhou [23] is among the information-theory-based graph summarization methods that use minimum description length (MDL) as the principle to find a summary graph while minimizing the total description length. The description has two terms, the summary graph and the errors or changes required to exactly reconstruct the original graph. The algorithm seeks to minimize the size of summary graphs and their reconstruction error. Like many MDL-based methods, DPGS uses a locality-sensitive hash (LSH) at each iterative stage to divide the current set of supernodes into disjoint groups, and performs greedy merging of the candidate group-member nodes. In DPGS, the merging cost is defined such that the effect is to consistently group nodes that have similar neighbors, a property that further lends to its suitability for graph clustering. The most unique feature of DPGS is its novel reconstruction approach, which informs both the second term used to reconstruct the graph exactly and the later process of reproduction of the graph. It is based on the configuration model that assigns superedges proportional to node degrees. The configuration model achieves lower reconstruction error of the reconstructed adjacency matrix than the commonly used scheme, uniform reconstruction, as measured by the generalized KL-divergence error, a type of Bregman divergence [4], [7]. Zhou shows theoretically that the reconstruction approach bounds the perturbation of the graph spectrum. The time complexity of DPGS is linear in the number of node edges, $\mathcal{O}(T \cdot |E|)$, where T is the number of iterations of the algorithm's main grouping-merge loop, which is set by the user and is set to 30 by default, and E is the set of graph edges.

III. Related Work

A summary of approaches to community detection or clustering can be found in Fortunato [6]. There are many methods, often with different objectives and different results. A general theory for objective functions for clustering is not yet well characterized and the field is still empirically driven. Many studies are still needed to understand the underlying mechanisms that are responsible for the interactions of nodes in different graphs, see for example [2], [6], [15], [19].

One community detection method that is particularly suited to SBM-based graphs is a sequential implementation of a parallelizable Bayesian statistics-based stochastic block partitioning method of Peixoto [8], [18]. SGC uses this method as its baseline algorithm. The baseline partitioner from 2017 is accurate in its group assignments, but slow: its time complexity order is $\mathcal{O}(|E| \cdot log^2|E|)$ where E is the number of edges in the graph.

One of the winners in the 2017 SGC competition is a partitioner by Zhuzhunashvili and Knyazev that has at its core the Locally Optimal Block Preconditioned Conjugate Gradient Method (LOBPCG) [24]. LOBPCG is a mature implementation of a spectral method particularly suited to sparse matrices and sparse graphs like those of the SGC [24]. It performs spectral clustering, which derives a partition from the eigen-decomposition of the graph Laplacian matrix. Zhuzhunashvili's partitioner has a faster time order than the baseline. It has a time order

of $\mathcal{O}(|N| \cdot k)$, where N is the set of nodes in the graph, and k is the number of groups, or clusters.

This algorithm uses the locally-optimal block partition conjugate gradient, a spectral clustering method developed by Knyazev in 2001 [12] that has been demonstrated as a significant improvement to the traditionally used Lanczos method. Zhuzhunashvili and Knyazev showed the utility of this spectral clustering method within the context of the SGC [24]. They reported very high partition quality, with 99% or better accuracy against the known truth partition, for both static and streaming graph partition with the Challenge graphs, for the 2017 SGC dataset of graphs of size 50,000 nodes to 2M nodes, and found a 100-1000$\times$ speedup over the baseline partitioner. They observed performance within the experimental set of 2017 static graphs in line with the time-order for the main underlying function, LOBPCG.

In our experiments using their techniques with the more challenging 2020 datasets, however, partitioning quality was seriously degraded with this approach—down to 40% label accuracy in many cases. We previously demonstrated a solution to this problem [5]. One important difference is the formulation of the graph Laplacian used. Our solution utilizes the normalized Laplacian $L = D^{-1/2}WD^{-1/2}$ in a search for the largest eigenvalues. Zhuzhunashvili used the nonnormalized Laplacian $L = D - W$ in search of the smallest eigenvalues.

This had similar ultimate accuracy to what Zhuzhunashvili and Knyazev reported, yet which also held for the more challenging 2020 graphs. It also had the benefit of converging to a near-optimal solution faster than reported for the nonnormalized Laplacian $L = D - W$, typically reaching its solution at around Stage 3, or 30% of the graph edge data. Similar to Zhuzhunashvili and Knyazev, we use the clusterQR method by Damle [1] to assign labels based on the eigenvectors. Also we set our initial guess of number of clusters $l > k$ and then run again with this output as the precondition matrix, with the new number of clusters k set to the first jump in values beyond the principle eigenvector. We refer to this approach as LOBPCG′, reflecting that it is Zhuzhunashvili and Knyazev's approach with some small but significant changes.

Graph summarization is a graph compression technique that can be used to reduce graph storage requirements or produce smaller input to other graph analysis. LeFevre and Terzi first proposed graph summarization based on the MDL principle [14]. Many improvements of the two-term encoding of the MDL have been developed by a number of researchers [13], [20]–[23]. Among these, DPGS stands out as having a formulation most closely aligned with the related but independent problem of determining and preserving community structure. As such, it appears most likely to exhibit good performance in the crossover task of community detection.

Zhou [23] evaluated the description length produced by DPGS for synthesized graphs generated from a random model with a power-law degree distribution, using power-law parameter $\alpha = -3.0, -3.5$, and -4.0, for graphs ranging in size from 500 nodes to 4,000 nodes. They also evaluated DPGS reconstruction error, for eight real-world graphs and for the F1-scores of the resulting summary graphs in GNNs. They also demonstrated linear scaling for the algorithm by running variously sized samples of their largest real-world dataset. They also provide several comparisons with the commonly used uniform reconstruction scheme vs the configuration model within algorithms `k-Gs` and `SSumM`.

Our work is useful to the graph summarization research community in that it extends Zhou's time performance assessment to a broader sample of graph sizes and known structural properties, including summarization of graphs with differing probability of overlapping, or mixing, between groups, and low to high group size variation, and for a larger range of graph sizes, ranging up to 5 million nodes. Additionally, the present work evaluates performance against a different problem, namely the suitability of this graph summarization technique as a community detection method, by assessing the quality of the communities generated by DPGS. This is achieved by scoring its supernode assignments against the expected assignments for graphs with known ground truth.

IV. Approach

The hypothesis tested here is that several design features of DPGS make it suitable as a community detection or clustering algorithm, in addition to its intended function as a graph sparsification / summarization technique. There are three aspects to suitability assessed here:

Partition quality: does the algorithm provide group labels to nodes from the original graph convenient to the community detection task at hand? Although the definition of communities is not available for SGC [8], the community detection task appears to be finding a grouping within sparse graphs that maximizes interconnectedness *within* groups while minimizing interconnectedness *between* groups. We define quality as being on par with that produced by the SGC baseline algorithm, or better.

Computational efficiency: does the partition occur within an acceptable amount of time, and require an acceptable amount of computing resources, such as memory? We define that here as being no slower than the baseline algorithm.

Practical barriers to use: does the approach require computing resources readily available to its intended users? For the purposes of this project we define the target hardware as commodity systems that are readily available to this project, no more expensive than a server-class machine, and requiring no commercial software licenses to conduct the experiments.

We assessed overall suitability of an algorithm to community detection by fixing the hardware resources across the board, then quantifying partition quality and computational efficiency for graphs from the SGC dataset that are of increasing size and varying structural composition. We did so using forty graphs of varying size and partitioning-task difficulty. We did the same for two alternative approaches—one, the baseline, and two, a mature/already-high-performing algorithm that is the current reigning champion. This helps to better understand the significance of the results.

V. Methods

The algorithm implementations compared here are: a) DPGS, the hard parts of which are implemented in C++ and the rest in Python; b) the SGC baseline, which is Kao's 2017 Python implementation of Peixoto's approach [8], and c) a Python implementation of LOBPCG′ that we describe in a previous publication [5]. This implementation depends on the implementation of `lobpcg` within the `scipy` Python scientific toolkit [11].

A. Performance Metrics

Algorithms are assessed along two orthogonal axes loosely construed as their cost in running time versus their benefit in output quality. Whereas in graph summarization this is judged as the description length of the output, here it is instead assessed as how well the algorithm found the latent structure within the graph, for graphs with known truth partitions.

For partition quality, we scored the supernode assignments in the final DPGS output using an array of metrics provided with the SGC that are based on the labels' commonalities with the expected group labels. In all cases the evaluation first finds the mapping between the algorithm's designated group labels and those of the known truth data.

SGC provides a comprehensive set of tools for assessing partition quality. This work computes all the SGC metrics described in Kao [8], in the convenient form implemented within the SGC baseline support functions. We report on only Accuracy here; however, these metrics include pairwise precision and recall, mutual information, and many other assessments. Accuracy is defined as the fraction of nodes correctly partitioned, which is the number matched of the algorithm's results with the true labels over the total number of nodes.

To achieve the goal of assessing algorithm computational efficiency, we tested the running time of the algorithm on a server-class machine with half a terabyte of memory that is characterized in Table I. Time was collected for a version of the code that minimized input and output (I/O), allowing merely input of the dataset and running of the algorithm, no write-out of output or informational messages and no evaluation of the results.

Time measurements reported here are true computational resource usage times, not elapsed wall clock time. Times are reported for the entire execution time, including loading and transforming inputs, and total execution time is $sys + user$, all time spent on the user's behalf either within the code or within the kernel reported by the Unix utility `time`.

In addition we evaluate the algorithms in terms of their speedup over the baseline. Speedup here is as is typically defined in terms of latency as t/W of the new algorithm over t/W of the old, where t is the total execution time and W is the workload.

Computational efficiency was also assessed as the size of the problem increases, i.e. as the graph size increases. This provides a measure of the scalability of the algorithm; and we did so more systematically than has been done before for DPGS by testing over a much larger size range of graphs, and with controlled variation of graph structure and task difficulty.

B. Datasets: Graphs with Known Group Labels

This work uses the SGC graph datasets described by Kao [8]. The synthetic SBM graphs that are integral to SGC have known partition label assignments, or *ground truth*. The dataset contains multiple sets of graphs differing in size but having similar characteristics to one another, such as probability of inter- and intra-cluster edges, maximum and minimum node degrees, as well as groups with similar block size, between-block interactions, and block size variation [3], [8]. Some have real-world graph embeddings within them as well. These do not contribute to the partition quality score. The

graphs range in size from 500 to 5M vertices. We worked with both the original set of graphs from 2017 and those released in 2020.

TABLE I
MACHINE PROPERTIES FOR EXPERIMENTATION

Memory size	512 GB
Cores	64
Processor	AMD Opteron 6376
Clock Frequency	2.3 GHz
Floating point	64-bit double precision
OS	CentOS Linux 7.9.2009
Python	3.6.9
scipy	1.5.0
gcc compiler	8.5.0
Boost library	1.77.0

C. Computing Resources

Experiments were carried out on a server-class machine with 512 GB of memory. The machine characteristics are summarized in Table I for both hardware and software. The large memory size allowed us to run LOBPCG′ against the 2 million- and 5 million-node graphs without needing to employ its more-complex block handling capabilities. This machine was unable to run the baseline algorithm in Python to completion on graphs larger than 50,000 nodes in a reasonable amount of time. It was able to run DPGS on graph sizes up to 1-2 million nodes. Graphs toward the upper end of an algorithm's capabilities generally took months of wall-clock time.

VI. Experiments & Results

The accuracy, time usage and time-complexity of each algorithm were evaluated using graphs with known partition solutions. Typically, three trials were performed for each experiment. Figure 1 shows the results for the three algorithms broken down by the fundamental structural classes of graphs within the dataset.

DPGS typically produces a good partition of the graph, roughly in line with the baseline algorithm's performance, producing around the same number of clusters as expected. Further, DPGS performance did not suffer much as the task difficulty increased. It often provides acceptably high performance for less structured graphs such as ones with a high degree of mixing between groups (sets *HL* and *HH*), and a high variation in group size (sets *LH* and *HH*). DPGS performance deteriorated a bit slower than that of the baseline as the task difficulty increased. For both the Peixoto-based baseline partitioner and DPGS, however, performance is not nearly as high as what we consistently observed for LOBPCG′.

Whereas Figure 1 reports on the results aggregated by graph type, Figure 3 provides more detail on the performance of DPGS as it relates to graph difficulty and graph size. The supernodes chosen by DPGS provide very high fidelity representation of the community structure for the graphs with low block overlap and low block size variation, which are the 2017 dataset and the *LL* graphs of various sizes from the 2020 dataset. Of the graphs that completed in a reasonable time, accuracy was typically 75% or better. The algorithm performance drops off precipitously, however, on the 200K node graph, especially for the case of both high block size variation and high overlap. The runs we performed for larger graphs in these sets *HL*, *LH* and *HH* have not completed yet in the two months we have been running them on the test machine. This performance dropoff in both time and partition quality is a point to be investigated further. It could be a limit within the algorithm, but it may be an artefact of the implementation rather than the algorithm—a result of parameter settings and of inherent constraints within the DPGS implementation, for example, given how much larger these graphs are than those for which it was originally implemented and demonstrated.

We used the default parameter settings for the DPGS algorithm except for the number of iterations, which

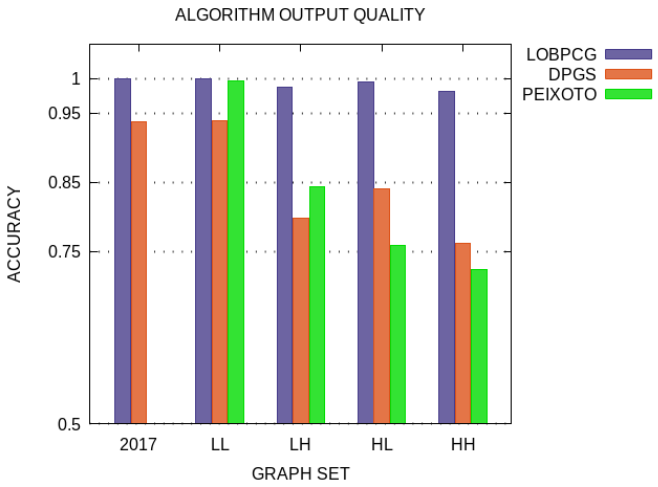

Fig. 1. Output quality for the three algorithms. Shown is average accuracy per class of graph. Task difficulty increases from left to right. Classes are the 2017 dataset, and the four kinds of graphs in the 2020 dataset: low overlap, low block size variation graphs (*LL*), low overlap, high block size variation graphs (*LH*), and so on. Only the runs that completed in a reasonable amount of time are presented.

we found defaulted to too low a value for these large graphs. We experimented with the ideal setting for *turns* for several graphs and arrived at a rough rule of thumb that $|N|/20$, where N is the number of nodes in the graph, provided the algorithm sufficient iterations to merge groups into roughly the number of clusters present in the labeled data. Runs used parameter settings of 8 hash bins, 42 as the random seed, and $|N|/20$ iterations.

We observed two outliers for which DPGS produced an undesirably large supernode graph in its MDL description of the graph. Of the two, one was brought into line with the predicted number of groups simply by specifying a larger number of turns. For the other, the 200K-node *HH* graph that is the lowest accuracy shown in Figure 3, we experimented with many possible parameter settings to try to improve the quality of the output, but the results appeared robust against parameter settings. In each case, the supernode graph showed little reduction, containing about 90% of the nodes of the original graph. Description length also was not reduced; instead, it was slightly larger than the original graph description. This implies that we were not seeing a result of graph summarization optimization.

The time complexity of DPGS was previously reported to be linear in the number of node edges, $\mathcal{O}(T \cdot |E|)$, where T is the number of group-merge iterations set by the user and E is the set of graph edges. We set T to $|N|/20$. All of this leads to the expectation that DPGS will produce runtimes in between the baseline and LOBPCG′. We would expect it to run typically

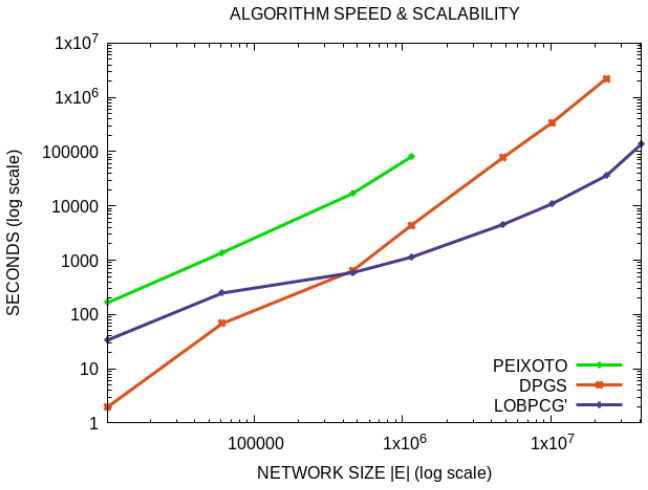

Fig. 2. Algorithm speed and scalability as graph size increases. Only those experiments that completed in a reasonable amount of time are shown. Both graph size and time used are displayed on a logarithmic scale. Time is defined in the section on Performance Metrics.

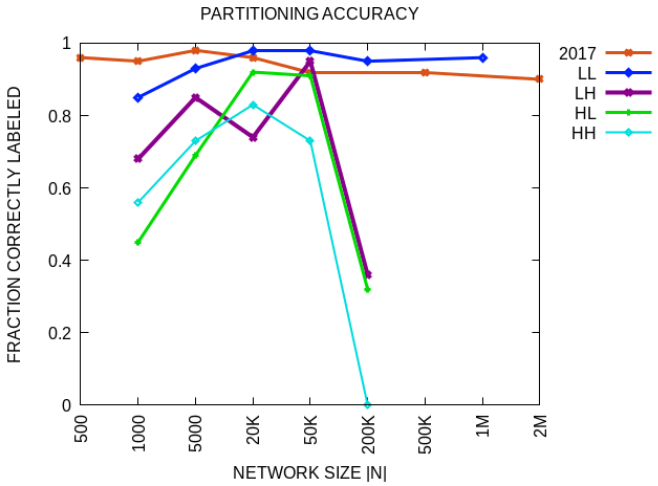

Fig. 3. DPGS algorithm accuracy further broken down for each graph in the 2020 dataset. Shown are the 2017 dataset, and the four kinds of graphs in the 2020 dataset: low overlap, low block size variation graphs (*LL*), low overlap, high block size variation graphs (*LH*), and so on. Accuracy exhibits a dropoff for the harder of the 200,000-node graphs. Runs for the larger sizes of (*LH, HL, and HH*) graphs took an unreasonable time to complete and are not shown.

significantly slower than LOBPCG′, which has time-order $O(|N| \cdot k)$ where k is the number of clusters and N is the set of graph nodes, but faster than Peixoto, which has time-order $O(|E| \cdot log^2|E|)$.

Figure 2 shows a log-log plot of the experimental results we observed on the test machine for the three algorithms. Timing results roughly correspond with the reported time orders of the algorithms. Both Peixoto and DPGS appear linear in the number of edges of the graph. LOBPCG′ is sublinear in $|E|$, with a larger constant factor for smaller graphs. More-complex graphs generally take longer for all three algorithms. We find that over the range of graph sizes that run within a few weeks, DPGS is consistently 20× faster than the DGC baseline across the range of graph sizes and typically an order of magnitude slower than LOBPCG′ except for very small graphs, for which DPGS is faster than LOBPCG′.

A. Additional Time Performance Assessments

For DPGS, the experiments use a Python implementation that calls the C++ compiled binary, whereas for Peixoto the algorithm we used is completely written in Python. This could bias the results toward DPGS for reasons other than algorithmic ones. This is also not the fairest comparison possible, given that C++ implementations exist for both Peixoto and LOBPCG that take

advantage of Boost acceleration and other specialized libraries.

We addressed this uncontrolled-for contribution to the timing results partially but not fully. For timing comparisons we also conducted experiments with the C++ version of the Peixoto algorithm. For these time performance runs, the C++ implementation of DPGS was compared against the Python implementations of the baseline and LOBPCG′. For instance, the baseline C++ code in the SGC github calls a function `minimize_blockmodel_dl` [17] which wraps the underlying function `multilevel_mcmc_sweep`. These are implemented within the inference library of `graph-tool` [16]. The function performs a user-specified number of multilevel agglomerative acceptance-rejection MCMC sweeps to sample network partitions and a bisection search on the number of groups along with group merges and singe-node moves. Although these functions are‘ callable inside Python, all the hard parts are implemented in C++. `Graph-tool` makes use of the Boost libraries and other optimized codes to run functions such as these efficiently.

We decided to try running the C++ Peixoto algorithm in its current 2021 form, within the latest release of `graph-tool`, which is Version 2.43. There were some practical barriers to this, however. After a week of effort installing `graph-tool` and all the dependencies on the target `CentOS` machine described in Table I without root access or a package manager, we were unable to run it without errors.

We succeeded in running these functions from `graph-tool` on a lesser machine for which we had the necessary accesses and resources (4-processor Intel i5-4670K CPU at 3.40GHz; 32 GB memory; Linux Hera 5.1.7). Despite being able to run the C++ version of the algorithm on this machine, the outputs did not correctly infer the modular graph structure. We learned that the current interface to the Peixoto library function did not allow setting some of the key parameters present in the SGC Python baseline, such as telling the algorithm to run multiple sweeps but stop when the entropy does not change (the maximum number of iterations and the stopping criterion ϵ). The degree-correctedness of the graph was buried another level or two further below the interface provided by `minimize_blockmodel_dl`. It may be that the 2017 version of the function had exposed these factors, but the contemporary version does not. We were unable to find parameter settings that resulted in a graph partition with fewer than $b \approx N$ blocks. Given this unacceptable output quality, a comparison with the C++ version of Peixoto's algorithm was not possible.

The remaining point of comparison available was to evaluate the constant factor time difference between one iteration of the block model inference function within `graph-tool` and the DGC baseline reference implementation in Python. This is useless, of course, to extrapolate a performance factor for the C++ version, unless one assumes it *could* be run with some settings that produced a good partition and that the underlying algorithm was still substantially the same as the one in the Python baseline. We assessed this constant factor difference for one iteration of each algorithm, after setting both to perform the MCMC sweep once, and then running both across the tractable sizes of graphs within the dataset, up to 50K nodes, for all graphs in the 2017 and 2020 datasets. This first MCMC sweep in the C++ version took on average a tenth of the cycle time of the Python code or $10\times$ Speedup, and the performance gap appeared to increase with graph size.

VII. Discussion & Future Work

This work provides more insight into the performance of a new graph summarization method, DPGS, by extending the set of graphs to which it has been applied, resulting in more empirical information about its compression ratio and runtime performance. It also evaluates the method in a different context from those explored before by evaluating its usage for community detection, or clustering. In doing so, it puts the results into the context of existing graph community detection methods and implementations by comparing DPGS's performance with prominent alternatives.

We apply a graph summarization method, DPGS, that lossily encodes the graph using fewer nodes and edges while maintaining key properties of the original such as the eigenspace of the graph Laplacian within a bounded error. The algorithm does so by a combination of design elements demonstrated and/or formally proven by Zhou, such as a) preferentially merging supernodes that share edges; b) capturing the skewed degrees of real-world graphs; and c) preserving the spectrum of the Laplacian of the original graph within a KL-divergence bound. While these are design features whose primary intended reason is to improve the graph reconstruction model for lossy MDL-based graph summarization, we show here that they also create a summarization that preserves the latent communities within the graph, for the DARPA/MIT Graph Challenge streaming stochastic block partition dataset, which are synthetic SBM graphs

with some hybridization with real-world graphs, with realistic power-law degree distributions in the range between -3 and -2, and known community labels.

We find that DPGS produces a supernode graph that preserves the community structure of the graphs in an order of magnitude less time than the Graph Challenge baseline partitioning method, at 20× Speedup across graphs ranging from 1,000 to 2 million nodes in size. Our results provide evidence that the first term of its output MDL summarization is a suitable more-compact representation of the community structure of the graph for similar graphs, It appears that the degree-preserving features of DPGS are also preserving the community structure of the graph as a whole. The summarization produced a representation of the original graph that averaged 75.4% of the original graph description length across the set of forty graphs.

Future work includes identifying the causes of the performance dropoff we witnessed in using DPGS for "difficult" graphs beyond a certain size, experimentation with the DPGS algorithm to further enhance its community detection capabilities and its ability to handle very large graphs, and an all-C++ time comparison, with a C++ implementation of LOBPCG and the baseline algorithm.

VIII. Acknowledgements

The authors would like to thank Andrei Knyazev and Houquan Zhou for helpful information and discussions and the anonymous reviewers for helpful comments. This research was supported in part by an appointment to the Intelligence Community Postdoctoral Research Fellowship Program at the National Institute of Standards and Technology (NIST), administered by Oak Ridge Institute for Science and Education (ORISE) through an interagency agreement between the U.S. Department of Energy and the Office of the Director of National Intelligence (ODNI).

References

[1] Anil Damle, Victor Minden, and Lexing Ying. Robust and efficient multi-way spectral clustering. *arXiv preprint arXiv:1609.08251*, 2016.

[2] Vinh Loc Dao, Cécile Bothorel, and Philippe Lenca. Community structure: A comparative evaluation of community detection methods. *Network Science*, 8(1):1–41, 2020.

[3] DARPA/MIT. Data sets | graphchallenge. *Graph Challenge Website https://graphchallenge.mit.edu/data-sets*, 2016.

[4] Inderjit S Dhillon and Suvrit Sra. Generalized nonnegative matrix approximations with bregman divergences. In *NIPS*, volume 18. Citeseer, 2005.

[5] L.J.K. Durbeck and P. Athanas. Incremental streaming graph partitioning. In *2020 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–8, 2020.

[6] Santo Fortunato. Community detection in graphs. *Physics reports*, 486(3-5):75–174, 2010.

[7] Keith Henderson, Brian Gallagher, Tina Eliassi-Rad, Hanghang Tong, Sugato Basu, Leman Akoglu, Danai Koutra, Christos Faloutsos, and Lei Li. Rolx: structural role extraction & mining in large graphs. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1231–1239, 2012.

[8] Edward Kao, Vijay Gadepally, Michael Hurley, Michael Jones, Jeremy Kepner, Sanjeev Mohindra, Paul Monticciolo, Albert Reuther, Siddharth Samsi, William Song, et al. Streaming graph challenge: Stochastic block partition. In *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–12. IEEE, 2017.

[9] Edward K Kao, Steven Thomas Smith, and Edoardo M Airoldi. Hybrid mixed-membership blockmodel for inference on realistic network interactions. *IEEE Transactions on Network Science and Engineering*, 6(3):336–350, 2018.

[10] Brian Karrer and Mark EJ Newman. Stochastic blockmodels and community structure in networks. *Physical review E*, 83(1):016107, 2011.

[11] A Knyazev. Locally optimal block preconditioned conjugate gradient method (lobpcg). *Scipy.org https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.linalg.lobpcg.html*, 2020.

[12] Andrew V Knyazev. Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method. *SIAM journal on scientific computing*, 23(2):517–541, 2001.

[13] Kyuhan Lee, Hyeonsoo Jo, Jihoon Ko, Sungsu Lim, and Kijung Shin. Ssumm: Sparse summarization of massive graphs. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 144–154, 2020.

[14] Kristen LeFevre and Evimaria Terzi. Grass: Graph structure summarization. In *Proceedings of the 2010 SIAM International Conference on Data Mining*, pages 454–465. SIAM, 2010.

[15] Leto Peel, Daniel B Larremore, and Aaron Clauset. The ground truth about metadata and community detection in networks. *Science advances*, 3(5):e1602548, 2017.

[16] Tiago Peixoto. Graph tool: Efficient network analysis. *Skewed.de http://graph-tool.skewed.de/*, 2021.

[17] Tiago Peixoto. Inferring modular network structure. *Skewed.de https://graph-tool.skewed.de/static/doc/demos/inference/inference.html#the-stochastic-block-model-sbm*, 2021.

[18] Tiago P Peixoto. Efficient monte carlo and greedy heuristic for the inference of stochastic block models. *Physical Review E*, 89(1):012804, 2014.

[19] Carey E Priebe, Youngser Park, Joshua T Vogelstein, John M Conroy, Vince Lyzinski, Minh Tang, Avanti Athreya, Joshua Cape, and Eric Bridgeford. On a two-truths phenomenon in spectral graph clustering. *Proceedings of the National Academy of Sciences*, 116(13):5995–6000, 2019.

[20] Matteo Riondato, David García-Soriano, and Francesco Bonchi. Graph summarization with quality guarantees. *Data mining and knowledge discovery*, 31(2):314–349, 2017.

[21] Kijung Shin, Amol Ghoting, Myunghwan Kim, and Hema Raghavan. Sweg: Lossless and lossy summarization of web-scale graphs. In *The World Wide Web Conference*, pages 1679–1690, 2019.

[22] Quinton Yong, Mahdi Hajiabadi, Venkatesh Srinivasan, and Alex Thomo. Efficient graph summarization using weighted lsh at billion-scale. In *Proceedings of the 2021 International Conference on Management of Data*, pages 2357–2365, 2021.

[23] Houquan Zhou, Shenghua Liu, Kyuhan Lee, Kijung Shin, Huawei Shen, and Xueqi Cheng. Dpgs: Degree-preserving graph summarization. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, pages 280–288. SIAM, 2021.

[24] David Zhuzhunashvili and Andrew Knyazev. Preconditioned spectral clustering for stochastic block partition streaming graph challenge (preliminary version at arxiv.). In *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–6. IEEE, 2017.